UNIUNEA EUROPEANĂ

GUVERNUL ROMÂNIEI

Instrumente Structurale
2007-2013

Platformă de e-learning și curriculă e-content pentru învățământul superior tehnic

Baze de date 1

13. Baze de date distribuite

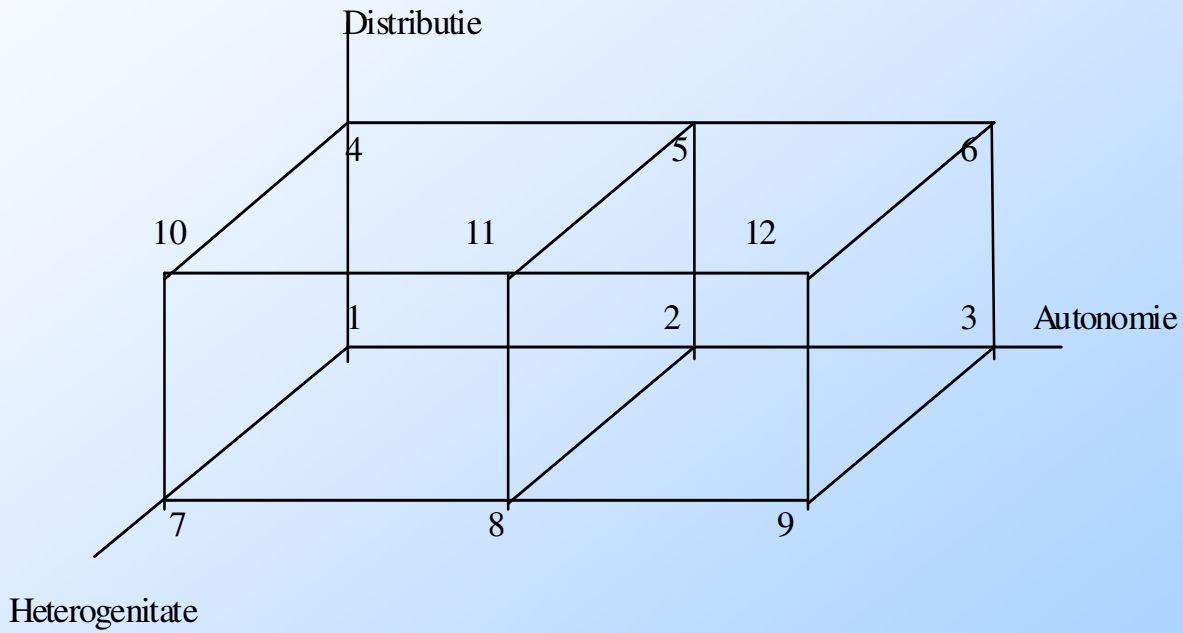
Definitii

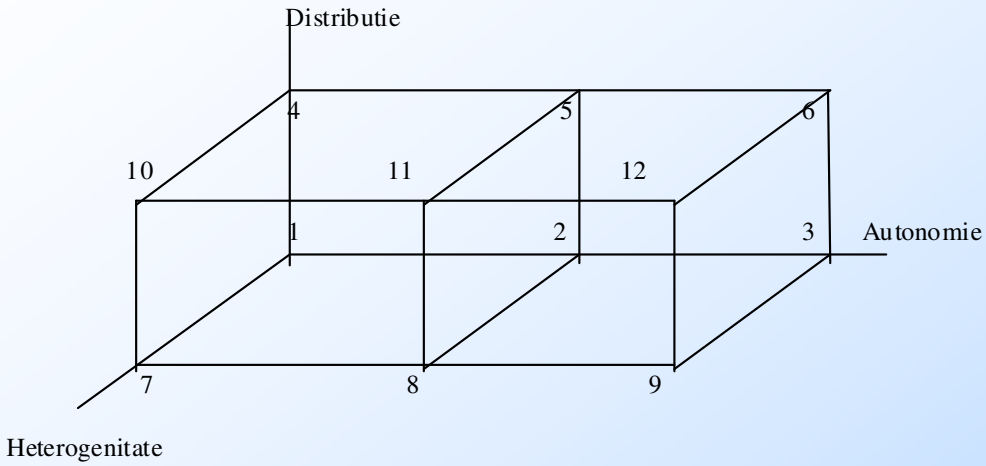
- ***Definitie***: **O baza de date distribuita (BDD)** este o colectie de baze de date logic interconectate distribuite intr-o retea de calculatoare.
- ***Definitie***: Un **SGBD distribuit** este format din colectia de programe care permit gestiunea bazelor de date distribuite facand distributia transparenta pentru utilizator.

Clasificarea SGBD distribuite

- Clasificarea SGBD distribuite se poate face avand in vedere trei caracteristici ale sistemelor aflate in nodurile locale ale sistemului:
  - Distributie
  - Autonomie
  - Heterogenitate
- In Ozsu, Tamar M; Valduriez, Patrick: *Principles of Distributed Database Systems*, Prentice Hall 1991 este prezentata urmatoarea clasificare a SGBD:

Clasificarea SGBD distribuite





1 = Sisteme omogene, nedistribuite si logic integrate
2 = Sisteme omogene, federate, nedistribuite
3 = Sisteme multibaza de date omogene nedistribuite
4 = Sisteme omogene distribuite
5 = Sisteme federate omogene
6 = Sisteme multibaza de date omogene
7 = Sisteme heterogene integrate
8 = Sisteme federate heterogene
9 = Sisteme multibaza de date heterogene
10 = Sisteme heterogene distribuite
11 = Sisteme federate heterogene distribuite
12 = Sisteme multibaza de date heterogene distribuite

Optimizarea executiei cererilor

- Prezentam in continuare o trecere in revista a problematicii generale a optimizarii cererilor in baze de date distribuite. Exista o serie de tehnici de optimizare care devin relevante doar in acest context. In cazul bazelor de date centralizate, un aspect foarte important in contextul optimizarii cererilor este minimizarea numarului de accese la disc.
- Sa luam exemplul (prezentat si in cartea lui J.D. Ullman) unui produs cartezian intre doua relatii de cate doua atribute, fie ele R si S avand *r* respectiv *s* tuple.
- Fie *nr* respectiv *ns* numarul de tuple din fiecare relatie care incap intr-un bloc pe disc, si *m* numarul de blocuri disponibile in memoria interna a calculatorului.
- Procesul de calcul se desfasoara astfel:
  - citim numarul maxim posibil de blocuri din S (adica *m-1* blocuri) si pentru fiecare inregistrare din R citim in intregime, in ultimul bloc disponibil, relatia S.
- Atunci:

Optimizarea executiei cererilor

- Numarul de blocuri citite pentru a parcurge relatia R este
  r/nr
- Numarul de parcurgeri al relatiei S este
  r/((m-1)*nr)
- Pentru o parcurgere a relatiei S se citesc
  s/ns blocuri
- Deci numarul de accese la disc este in total:
  (r/nr) * (1 + s/((m-1) * ns))
- Ceea ce poate duce, in cazul relatiilor cu numar mare de tuple, la timpi mari pentru executia unei astfel de cereri.
- In cazul de mai sus, deoarece numarul de accese este mai puternic influentat de r/nr decat de s/ns, vom lua ca relatie R pe cea care are acest raport mai mic.

Optimizarea executiei cererilor

- In implementarea joinurilor, se folosesc de asemenea diverse metode:
  - sortarea uneia dintre relatii dupa atributele implicate in join (in cazul echijoinului)
  - folosirea indecsilor (daca exista) pe cimpurile implicate in join, pentru acces direct la tuplele care dau elemente ale rezultatului.
  - micsorarea numarului de tuple luate in calcul, prin aplicarea mai intii a selectiilor, daca acestea sunt prezente in cerere.

Micsorarea numarului de accese la disc

- In literatura de specialitate sunt descrise o serie de principii (strategii) care duc la micsorarea numarului de accese la disc:

1. Realizarea selectiilor cat mai devreme posibil.
2. Combinarea anumitor selectii cu produse carteziene adiacente pentru a forma un join.
3. Combinarea secventelor de operatii unare (selectii si proiectii) intr-una singura (o selectie sau/si o proiectie)
4. Cautarea subexpresiilor comune, pentru a fi evaluate o singura data.
5. Folosirea indecsilor sau sortarea relatiilor, daca se obtine o crestere a performantelor.
6. Evaluarea diverselor strategii posibile inainte de a incepe procesul de calcul efectiv (in cazul in care sunt posibile mai multe metode de calcul) pentru a alege pe cea mai eficienta.

Aspecte ale optimizarii cererilor in BDD

- In cele ce urmeaza vom considera ca o baza de date consta dintr-o serie de noduri, fiecare fiind un calculator avand facilitati de stocare permanenta a datelor.
- In fiecare nod exista un server de tranzactii care prelucreaza cererile utilizatorilor si un server de fisiere care gestioneaza datele stocate.
- Datele sunt memorate sub forma de articole (items), articolul fiind unitatea atomica posibil de blocat (lock).

Aspecte ale optimizarii cererilor in BDD

- Este posibil ca anumite articole sa fie duplicate, existand in mai multe noduri, pentru cresterea vitezei de evaluare a cererilor(in literatura de specialitate acest fapt se numeste replicare).
- Nodurile sunt conectate prin legaturi care asigura circulatia mesajelor intre acestea. Costurile (ca timp) de evaluare a cererilor vor fi invers proportionale cu viteza asigurata de aceste legaturi.

Obiectivele optimizarii

- In literatura de specialitate sunt mentionate doua functii obiectiv pe care optimizarea trebuie sa le minimizeze:
  - **Costul total** = suma timpilor necesari procesarii cererii in nodurile implicate de catre aceasta plus timpii de comunicatie.
  - **Timp de raspuns**: intervalul de timp in care este evaluata cererea. Acesta este in general mai mic decat costul total datorita in principal paralelismului in evaluarea cererii (*paralelism intra-query*).

Obiectivele optimizarii

- Din punct de vedere al momentului cand are loc optimizarea, aceasta poate fi:
  - **Statica**: in momentul compilarii cererii.
  - **Dinamica**: in momentul executiei cererii.
  - **Euristica**: initial are lor o optimizare statica iar la momentul executiei cererii are loc o optimizare dinamica in cazul in care dimensiunea rezultatelor intermediare este mult diferita de cele anticipate.

Obiectivele optimizarii

- Toate aceste tipuri de optimizare necesita mentinerea de statistici asupra dimensiunilor relatiilor si fragmentelor precum si a rezultatelor operatorilor relationali asupra lor.
- Exista sisteme in care optimizarea se face *complet* in sensul ca se testeaza toate strategiile de evaluare si se adopta cea mai buna sau *euristic* in cazul in care se cauta un optim local, fara a se evalua toate posibilitatile de procesare a cererii.

Obiectivele optimizarii

- Optimizarea trebuie sa ia in considerare si topologia retelei: in cazul retelelor de tip WAN costurile de comunicatie pot fi importante pe cand la retelele locale (LAN) acestea se pot uneori neglija.
- Replicarea este si ea importanta: ea duce la scaderea costului de evaluare a cererilor de regasire in baza de date prin selectarea copiei cu costul de acces cel mai mic insa creste timpul de actualizare a bazei de date din cauza necesitatii actualizarii tuturor copiilor unei relatii.

Fragmentarea relatiilor

- Distribuirea datelor in BDD se face prin fragmentarea relatiilor logice (inexistente fizic) in fragmente existente pe diverse calculatoare.
- Fragmentarea se poate face pe orizontala, in care caz reconstituirea relatiei logice se face prin operatorul relational de reuniune, sau pe verticala in care caz reconstituirea se face prin join natural.

1. O relatie (logica) R poate fi joinul natural al mai multor fragmente:

- R = R1 |X| R2 |X| ... |X| Rn
- Daca ne reprezentam R ca pe o tabela, Ri vor fi multimi de coloane din aceasta, numite fragmente verticale.

Fragmentarea relatiilor

- continuare

2. O relatie R (logica) poate fi reuniunea mai multor fragmente:

- $R = R1 \cup R2 \cup ... \cup Rn$

- Daca ne reprezentam R ca pe o tabela, Ri vor fi multimi de linii din aceasta, numite fragmente orizontale.

Optimizarea costurilor de transmisie prin semijoinuri

- Sa consideram doua relatii R si S implicate intr-un join natural, relatiile fiind alocate in noduri diferite.
- Metoda cea mai simpla (dar nu si cea mai rapida) de calcul a joinului este de a trimite o copie a unei relatii in celalalt nod, calculul facandu-se acolo.
- Costul transmisiei este in acest caz:

c0 + min(r, s)

unde c0 este costul initierii legaturii iar min(r, s) costul transmisiei unei relatii (cea mai mica) in celalalt nod.

Optimizarea costurilor de transmisie prin semijoinuri

- Sa vedem insa o alta metoda pentru a efectua aceeasi operatie, folosind operatorul relational de semijoin, definit mai sus.
- O proprietate a acestui operator este ca

$$R \mid X\ S = R \mid X\ \pi_{R \cap S}(S)$$

ceea ce face posibila urmatoarea metoda de a calcula, optimizand costul transmisiei de date, joinul natural R|X|S:

1. Se trimite $\pi_{R \cap S}(S)$ in nodul R.
2. Se calculeaza R |X S in nodul R
3. Se trimite R |X S in nodul S
4. Se calculeaza R |X| S = (R |X S) |X| S

Utilizare semijoinuri

- Daca notam:
  - c0: costul initierii legaturii.
  - r1, s1: proiectiile lui R, respectiv S, pe $R \cap S$
  - r2, s2: dimensiunile lui R |X S respectiv R X| S
- atunci costul etapelor 1 si 3 de mai sus este:
  - 1. c0 + s1
  - 3. c0 + r2
- Deci, in total 2*c0 + s1 + r2.

Utilizare semijoinuri

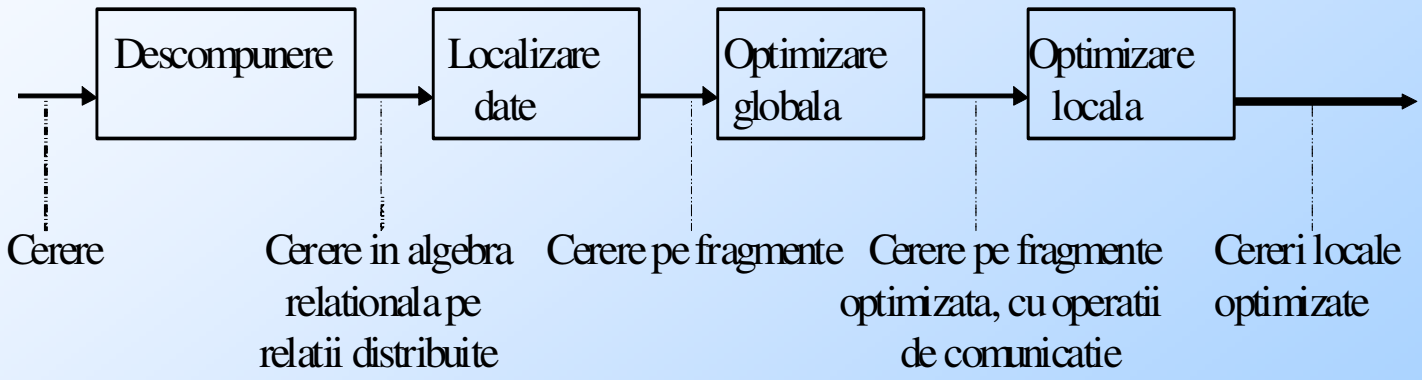
- In cazul in care c0 + min(s1+r2, s2+r1) < min(r,s) aceasta strategie este mai ieftina din punct de vedere al transmisiei decat cea descrisa la inceputul paragrafului.
- In cele ce urmeaza vom numi reducerea lui Ri in raport cu R1, R2, ..., Rn proiectia joinului celor n relatii pe Ri:

$$red(Ri) = \pi_{Ri}(R1 \,|X|\, R2 \,|X|\, \ldots \,|X|\, Rn)$$

Etapele procesarii unei cereri distribuite

1. Dupa cum se vede din diagrama urmatoare, cererea este intai analizata si descompusa in subcereri pe fragmente.
2. Urmatoarea etapa este localizarea fragmentelor implicate in evaluarea cererii.
3. Urmeaza o optimizare globala a cererii, in care pentru cererile pe fragmente rezultate din pasul anterior se cauta o strategie cat mai buna de evaluare.
4. Ultima etapa este optimizarea locala a fiecarei subcereri, in functie de caracteristicile nodului in care se evalueaza.

Etapele procesarii unei cereri distribuite



Proiectarea distributiei datelor

- Asa cum am vazut, caracterizarea unui sistem de gestiune a bazelor de date distribuite se face incadrandu-l dupa cele trei axe reprezentand caracteristicile de implementare ale acestuia:
  a. Distributie
  b. Autonomie
  c. Heterogenitate

- In functie de caracteristicile sistemului, de pozitionarea sa in spatiul acestor trei dimensiuni, se face si proiectarea distributiei datelor, alegand metoda cea mai potrivita cu arhitectura sistemului.
- Exista mai multe abordari in proiectarea unei baze de date distribuite:

Top-down

- **Strategia top-down** este folosita mai ales in cazul proiectarii unor noi aplicatii in care gradul de autonomie este scazut si omogenitatea este ridicata.
- Aceasta strategie este adecvata de exemplu in proiectarea sistemelor distribuite omogene.

Bottom-up

- **Strategia bottom-up** este folosita in cazurile in care gradul de autonomie este mediu sau ridicat si in cazul neomogenitatii sistemelor locale, prezente in diversele noduri.
- Un exemplu de situatie in care aceasta strategie este adecvata este in cazul in care bazele de date din nodurile locale exista deja, urmand a se face o integrare a lor intr-o baza de date distribuita, ramanand insa posibila si folosirea intr-o mai mica sau mai mare masura a fiecarei BD locale independent, ca o baza de date centralizata.
- Strategia se foloseste si in cazul sistemelor federate, multibaza de date si heterogene noi.

Strategia mixta

- **Strategia mixta**. Despre nici o strategie utilizata in practica pentru proiectarea bazelor de date distribuite nu se poate spune ca se incadreaza perfect intr-una din cele doua tipuri de mai sus.
- Insasi prezenta feedback-ului din strategia top-down este o abatere de la fluxul "de sus in jos" care trebuie sa guverneze aceasta strategie.
- De asemenea, in cazul strategiei bottom-up, bazele de date locale pot fi proiectate top-down, inainte de integrarea lor bottom-up.
- Exista metode de proiectare a bazelor de date distribuite pentru cazul in care informatia din noduri are o structura similara, metoda care imbina atat abordarea top-down cat si abordarea bottom-up.

Replicarea datelor

- Costul de operare in cazul unei BDD are doua componente principale:
  - costul de stocare a informatiei (numarul de accese la disc).
  - costul de comunicatie, pentru cereri si actualizari ale bazei de date.
- O redundanta mare duce la scaderea costului de comunicatie pentru cereri, deoarece probabilitatea existentei unei copii in nodul local este mare, in schimb creste costul de comunicatie pentru actualizarea tuturor copiilor.
- Redundanta scazuta are un efect contrar: scade costul de comunicatie pentru actualizari dar creste cel pentru cereri.

Redundanta optima

- Raportul frecventei actualizarilor, **r**, este un parametru cheie din cauza faptului ca un numar mare de copii si un procent mare de actualizari duce la o crestere a traficului de mesaje in retea, deoarece fiecare actualizare trebuie executata pe fiecare copie a unui fisier de date.
- Reversul este ca un procent mic de actualizari, cuplat cu un numar mic de copii (redundanta scazuta) poate avea ca efect un timp de raspuns mare pentru celalalt tip de operatii executate in baza de date (cautarile, cererile), procentul lor fiind mare, si anume **(1 - r).**

- Concluzia unui studiu prezentat in literatura de specialitate este ca o singura copie este suficienta in cele mai multe dintre cazuri si este dezavantajos sa existe mai mult decat cateva copii in cazul in care frecventa cererilor nu este cu mult mai mare decat cea a actualizarilor.

Redundanta optima

- Pentru realizarea acestui studiu a fost formulata problema si realizata o simulare a 80 de cazuri distincte pe trei tipuri de retele de comunicatie.
- Valorile raportate reprezinta medii ale rezultatelor obtinute.
- Desi nu este o rezolvare exacta ci mai degraba o inductie incompleta, numarul mare de cazuri luat in consideratie si diversitatea tipurilor de retele folosite dau rezultatelor valoare de adevar ridicata.
- De asemenea au fost efectuate calcule luandu-se in considerare reteaua ARPA din SUA si rezultatele obtinute au concordat cu alte rezultate bazate pe metode mai elaborate.

Redundanta optima

- Pentru realizarea studiului s-a pornit de la urmatoarele asertiuni:

1. Costul de stocare al datelor (una sau mai multe copii) este ignorat.
2. In sistem exista un original pentru fiecare fisier de date (fragment), copii ale acestuia putand exista in orice alt nod al sistemului.
3. Nu exista nici o restrictie asupra capacitatii canalelor de comunicatie.

- Toate aceste asertiuni sunt favorabile unei redundante ridicate: in ceea ce priveste prima asertiune, este evident. In ceea ce priveste asertiunile 2 si 3, ele duc la o subestimare a costului de comunicatie, deci de asemenea sunt favorabile unei redundante ridicate.
- De asemenea, problema consistentei diverselor copii nu este luata in considerare in modelul simplificat de la care se porneste. Acest lucru de asemenea favorizeaza cresterea redundantei.
- Rezulta ca, daca pornind de la aceste asertiuni, vom obtine totusi un numar mic de copii ca optim al redundantei fisierelor, concluzia enuntata mai sus este cu atat mai mult valabila.

Initializarea, recuperarea dupa incident si salvarea datelor

Definitii (Attar, Rony; Bernstein, Philip; Goodman, Nathan: *Site initialization, Recovery and Backup in a Distributed Database System,* IEEE Trans. On Soft. Eng., no 6, Nov. 1984)

- **Initializarea unui nod** (site initialization) inseamna integrarea unui nou nod intr-un sistem de baze de date distribuite.
- **Recuperarea dupa incident a unui nod** (site recovery) inseamna reintegrarea unui nod intr-un sistem de baze de date distribuit, atunci cand nodul a fost temporar avariat (software sau hardware).
- **Salvarea datelor** (site backup) reprezinta crearea unei copii statice a datelor unui nod (cu datele asa cum erau in momentul salvarii, dar intr-o stare consistenta) pentru arhivare sau alte motive.

Recuperare

- Asa cum am aratat, recuperarea dupa incident inseamna reintegrarea unui nod in sistemul distribuit dupa un timp in care acest nod a fost indisponibil. Problema este de asemenea ca orice copie x a unui articol X, copie care eventual a "ramas in urma" in urma opririi o perioada de timp a nodului, sa ajunga in concordanta cu celelalte copii din sistem.
- Asa cum se poate constata, recuperarea si initializarea sunt situatii aproape identice. Algoritmul descris mai sus poate fi aplicat din nou. In plus, in cazul recuperarii, se poate face o optimizare: doar acele articole X care au fost scrise dupa oprirea nodului in cauza trebuiesc recuperate. Toate celelalte articole, nemodificate in perioada de oprire, sunt accesibile sistemului de la inceput.
- Pentru a pune in practica aceasta optimizare, se pot folosi diverse metode. Una dintre ele este inregistrarea cererilor de scriere sosite in nodul de recuperat (asta inseamna ca in nodul respectiv continua sa ruleze un modul care monitorizeaza aceste cereri, lucru imposibil in cazul unui incident hardware). In cazul acesta, tranzactiile de copiere se vor executa doar pentru aceste articole, in orice ordine.

Salvare

- Problema salvarii datelor se pune din doua cauze: pentru arhivare si pentru cresterea vitezei unor cereri.
- In primul caz, copia se creeaza pentru a pastra pe suporturi permanente o copie a datelor, din motive de siguranta.
- In cazul al doilea, este evident ca, din cauza faptului ca o data creata o copie a datelor, aceasta nu se mai modifica. Cererile vor da deci rezultate specifice momentului in care a fost facuta aceasta. Reversul este ca deoarece aceasta copie nu este decat citita (nu si actualizata), si ea exista intr-un singur nod, datele pot fi accesate din nodul respectiv cu viteza mai mare (nu se pune problema concurentei in cazul copiilor read-only si nu este implicata in nici un fel partea de comunicatie de date intre noduri, operatiile fiind locale).

Salvare

- Pentru realizarea unei copii se poate de asemenea aplica algoritmul descris anterior: putem considera copia ca fiind un nou nod. Pentru a putea ingheta la o anumita stare (consistenta insa) aceasta copie, putem rula o tranzactie care blocheaza pentru citire toate datele care se doresc copiate. Aceasta tranzactie se va lansa dupa ce toate articolele din noul nod (virtual) au fost scrise cel putin o data.
- In momentul in care tranzactia de mai sus a terminat de efectuat blocarea, noul nod este scos din sistem. Datele sunt intr-o stare consistenta si pot din acest moment sa fie consultate fara nici o restrictie.

Bibliografie

- **[AtBeGo 84]** Attar, Rony; Bernstein, Philip; Goodman, Nathan: *Site initialization, Recovery and Backup in a Distributed Database System,* IEEE Trans. On Soft. Eng., no 6, Nov. 1984.
- **[ApHeYa 83]** Apers, Peter; Hevner, Alan; Yao, Bing*: Optimization Algorithms for Distributed Queries*, IEEE Trans. On Soft. Eng., no 1, January 1983.
- **[Ca.. 86]** Carey, M., DeWitt, D., Richardson, J., Sheika, E.: *Object and File Management in the EXODUS Extensible Database System*, Proceedings of the 12th International Conference on Very Large Data Bases, 1986.
- **[Ca.. 94]** Carey, M, DeWitt, D, Franklin, M., Hall, N., si altii*: Shoring Up Persistent Applications*, Proceedings of the ACM-SIGMOD Conference on the Management of Data, May 1994.
- **[CeNaWi 83]** Ceri, Stefano; Navathe, Shamkant; Wiederhold, Gio: *Distribution Design of Logical Database Schemas*, IEEE Trans. On Soft. Eng., no 4, July 1983.
- **[ChGr 85]** Chan, Arvola; Gray, Robert: *Implementing Distributed Read-Only Transactions*, IEEE Trans. On Soft. Eng., no 2, February 1985.
- **[ChHu 82]** Chu, Wesley; Hurley, Paul: *Optimal Query Processing for Distributed Database Systems*, IEEE Trans. On Soft. Eng., no 9, September 1982.
- **[ChSh 94]** Chaudhuri, Surajit; Shim, Kyuseok: *Including Group-By in Query Optimization*, Proc. of the 20th Intl. Conference on Very Large Databases, Santiago de Chile, 1994.
- **[Co 70]** Codd, E.F.: *A relational model for large shared data banks*, Comm. ACM, no. 6, 1970.
- **[CoGePl 81]** Coffman, Edward; Gelenbe, Erol; Plateau, Brigitte: *Optimization of the Number of Copies in a Distributed Data Base*, IEEE Trans. On Soft. Eng., no 1, January 1981.
- **[DeAd 88]** Delobel, Claude; Adiba, Michael: *Bases de donnees et systemes relationnels*, Dunod, 1988.
- **[Di 86]** Diener, Andreas Rudolf: *An Architecture for Distributed Databases on Workstations*, dissertation for the degree of Doctor of Technical Sciences, Swiss Federal Institute of Technology (ETH) Zurich, Diss ETH No 8088, 1986.
- **[ElNa 94]** Elmasri, R.; Navathe, S. : *Fundamentals of Database Systems*, The Benjamin/Cummings Publishing Company Inc., 1994
- **[Er 78]** Erlenkotter, D: *A dual-based procedure for uncapacitated facility location*, Oper. Res., vol. 29, Nov.-Dec. 1978.
- **[Fr.. 92]** Franklin, M., Zwilling, M., Tan, C.M., Carey, M., DeWitt, D.: *Crash Recovery in Client-Server EXODUS*, ACM SIGMOD 6/92, 1992.
- **[GaVa 85]** Gardarin, G., Valduriez, P.: *Bases de donnes relationelles*, Eyrolles, 1985.
- **[Gh 76]** Ghosh, S.P.: *Distributing a database with logical associations on computer network for parallel searching,* IEEE Trans. On Soft. Eng., vol SE-2, June 1976.
- **[HoRa 82]** Go, Gary; Ramamoorthy, C.V.: *Protocols for Deadlock Detection in Distributed Systems*, IEEE Trans. On Soft. Eng., no 6, Nov. 1982.
- **[JaKo 84]** Jarke, M., Koch, J.: *Query Optimization in Database Systems*, ACM Computing Surveys, nr. 2, 1984.
- **[MaPeSc 85]** Martella, Giancarlo; Pernici, Barbara; Schreiber, Fabio: *An Availability Model for Distributed Transaction Systems*, IEEE Trans. On Soft. Eng., no 5, May 1985.
- **[Mi 88]** Miranda, Serge: *Comprendre et concevoir les bases de donnees relationnelles*, Editests, Paris, 1988.
- **[Mu 92]** Muralikrishna, M.: *Improved Unnesting Algorithms for Join Aggregate SQL Queries*, Proc. of the 18th Intl. Conference on Very Large Databases, Vancouver, 1992.
- **[MuIbMiHa 85]** Muro, Shojiro; Ibaraki, Toshihide; Miyajima, Hidehiro; Hasegava, Tishiharu: *Evaluation of the File Redundancy in Distributed Database Systems*, IEEE Trans. On Soft. Eng., no 2, February 1985.
- **[OzVa 91]** Ozsu, Tamar M; Valduriez, Patrick: *Principles of Distributed Database Systems*, Prentice Hall 1991
- **[Pa.. 83]** Parker, Scott; Popek, Gerald; et.al.: *Detection of Mutual Inconsistency in Distributed Systems*, IEEE Trans. On Soft. Eng., no 3, May 1983.
- **[SiNa 85]** Sinha, Mukul; Natarajan, N.: *A Priority Based Distributed Deadlock Detection Algorithm*, IEEE Trans. On Soft. Eng., no 1, January 1985.
- **[SkSt 83]** Skeen, Dale; Stonebraker, Michael: *A Formal Model of Crash Recovery in a Distributed System*, IEEE Trans. On Soft. Eng., no 3, May 1983.
- **[SrSa 81]** Srinivasan, B.; Sankar, R.: *Algorithms to Distribute a Database for Parallel Searching, IEEE Trans.* On Soft. Eng., no 1, January 1981.
- **[TeYaFr 86]** Teorey, T; Yang, D; Fry, J: *A Logical Design Methodology for Relational Databases Using Extended Entity-Relationship Model*, ACM Computing Surveys, vol. 18, no. 2, June 1986
- **[Wo 83]** Wong, Eugene: *Dynamic Rematerialization: Processing Distributed Queries Using Redundant Data*, IEEE Trans. On Soft. Eng., no 3, May 1983.